2do Parcial – 2019s2 - Estructuras de Datos – UNQ

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todas las funciones y tipos de datos vistos en la práctica y en la teórica, salvo que el enunciado indique lo contrario.*
- *No se olvide de poner nombre, nro. de hoja y cantidad total de hojas en cada una de las hojas.*

Escuela de Magia

El objetivo de este examen es modelar una escuela de magos. Para ello, definiremos un tipo abstracto llamado `EscuelaDeMagia`. Y damos por hecho que:

- Existe un tipo abstracto llamado `Mago`, ya implementado, cuya interfaz se adjunta en el anexo de interfaces. Un mago puede aprende hechizos, e informarnos su nombre y qué hechizos conoce.
- El tipo `Hechizo` es sinónimo de `String`, y se corresponde con el nombre de un hechizo.
- El tipo `Nombre` es sinónimo de `String`, y se corresponde con el nombre de un mago.
- Podemos suponer que dos magos son iguales si poseen el mismo nombre, y un mago es más poderoso que otro si conoce más hechizos (lo que permite ordenarlos por la cantidad de hechizos que saben).
- En la escuela no existen dos magos con el mismo nombre.

Representación

Dicho esto, la representación que utilizaremos será la siguiente (*que no es posible modificar*):

```
data EscuelaDeMagia = EDM (Set Hechizo) (Map Nombre Mago) (PriorityQueue Mago)
```

Esta representación utiliza:

- Un `Set` de hechizos que la escuela ha enseñado a lo largo de su historia.
- Un `Map` que asocia magos con su respectivo nombre.
- Una estructura llamada `PriorityQueue` que posee a todos los magos de la escuela y permite obtenerlos de forma eficiente de mayor a menor en base a la cantidad de hechizos que saben.

Ejercicios

Invariantes

a) Dar invariantes de representación válidos según la descripción de la estructura.

Implementación

Implementar la siguiente interfaz de `EscuelaDeMagia`, utilizando la representación y los costos dados, calculando los costos de cada subtarea, y siendo M la cantidad de magos y H la cantidad de hechizos:

b) `fundarEscuela :: EscuelaDeMagia`
Propósito: Devuelve una escuela vacía.
Eficiencia: $O(1)$

c) `estaVacia :: EscuelaDeMagia -> Bool`
Propósito: Indica si la escuela está vacía.
Eficiencia: $O(1)$

d) `registrar :: Nombre -> EscuelaDeMagia -> EscuelaDeMagia`
Propósito: Incorpora un mago a la escuela (si ya existe no hace nada).
Eficiencia: $O(\log M)$

e) `magos :: EscuelaDeMagia -> [Nombre]`
Propósito: Devuelve los nombres de los magos registrados en la escuela.
Eficiencia: $O(M)$

f) `hechizosDe :: Nombre -> EscuelaDeMagia -> Set Hechizo`
Propósito: Devuelve los hechizos que conoce un mago dado.
Precondición: Existe un mago con dicho nombre.
Eficiencia: $O(\log M)$

g) `leFaltanAprender :: Nombre -> EscuelaDeMagia -> Int`
Propósito: Dado un mago, indica la cantidad de hechizos que la escuela ha dado y él no sabe.
Precondición: Existe un mago con dicho nombre.
Eficiencia: $O(\log M)$

h) `egresarUno :: EscuelaDeMagia -> (Mago, EscuelaDeMagia)`
Propósito: Devuelve el mago que más hechizos sabe y la escuela sin dicho mago.
Precondición: Hay al menos un mago.
Eficiencia: $O(\log M)$

i) `enseñar :: Hechizo -> Nombre -> EscuelaDeMagia -> EscuelaDeMagia`
Propósito: Enseña un hechizo a un mago existente, y si el hechizo no existe en la escuela es incorporado a la misma.
Nota: No importa si el mago ya conoce el hechizo dado.
Precondición: Existe un mago con dicho nombre.
Eficiencia: $O(M \log M + \log H)$

Usuario

Implementar las siguientes funciones **como usuario** del tipo `EscuelaDeMagia`:

j) `hechizosAprendidos :: EscuelaDeMagia -> Set Hechizo`
Propósito: Retorna todos los hechizos aprendidos por los magos.
Eficiencia: $O(M * (\log M + H \log H))$

k) `hayUnExperto :: EscuelaDeMagia -> Bool`
Propósito: Indica si existe un mago que sabe todos los hechizos enseñados por la escuela.
Eficiencia: $O(\log M)$

l) `egresarExpertos :: EscuelaDeMagia -> ([Mago], EscuelaDeMagia)`
Propósito: Devuelve un par con la lista de magos que saben todos los hechizos dados por la escuela y la escuela sin dichos magos.
Eficiencia: $O(M \log M)$

Bonus

m) Dar una posible representación para el tipo `Mago`, de manera de que se pueda cumplir con el orden dado para cada operación de la interfaz, pero sin implementarlas.

Anexo de interfaces

`Mago`, siendo H la cantidad de hechizos que sabe:

`crearM :: Nombre -> Mago`	$O(1)$
`nombre :: Mago -> Nombre`	$O(1)$
`aprender :: Hechizo -> Mago -> Mago`	$O(\log H)$
`hechizos :: Mago -> Set Hechizo`	$O(1)$

`Set`, siendo N la cantidad de elementos del conjunto:

`emptyS :: Set a`	$O(1)$
`addS :: Ord a => a -> Set a -> Set a`	$O(\log N)$
`belongsS :: Ord a => a -> Set a -> Bool`	$O(\log N)$
`unionS :: Ord a => Set a -> Set a -> Set a`	$O(N \log N)$
`sizeS :: Set a -> Int`	$O(1)$

`PriorityQueue`, siendo M la cantidad de elementos en la estructura:

`emptyPQ :: PriorityQueue a`	$O(1)$
`isEmptyPQ :: PriorityQueue a -> Bool`	$O(1)$
`insertPQ :: Ord a => a -> PriorityQueue a -> PriorityQueue a`	$O(\log M)$
`maxPQ :: PriorityQueue a -> a`	$O(1)$
`deleteMaxPQ :: Ord a => PriorityQueue a -> PriorityQueue a`	$O(\log M)$

`Map`, siendo K la cantidad de claves distintas en el map:

`emptyM :: Map k v`	$O(1)$
`assocM :: Ord k => k -> v -> Map k v -> Map k v`	$O(\log K)$
`lookupM :: Ord k => k -> Map k v -> Maybe v`	$O(\log K)$
`deleteM :: Ord k => k -> Map k v -> Map k v`	$O(\log K)$
`domM :: Map k v -> [k]`	$O(K)$